# SQL PRETRAINING SESSION 1 V1: PORTFOLIO HEALTH FOUNDATIONS

**Subtitle:** A Beginner-Friendly Guide with SQL Queries, Business Logic, and Output Screenshots

**Prepared By:** Your Name (Optional)

**Date:** August 2025

**Context:** This pretraining document is part of the Skill AI Path Data Analyst Track.

It covers foundational SQL concepts, query patterns, and business logic relevant for Google, loan portfolio health during crisis situations.

All queries are structured and explained with space for visual outputs and business interpretation.

**Organization:** Skill AI Path | EduFin Training Program

# TABLE OF CONTENTS

# 1. SESSION OVERVIEW

**Session Focus**

**Portfolio Health Assessment**

*"What is the scale and scope of our crisis?"*

This session introduces you to analyzing a loan portfolio in crisis using real-world SQL queries. You'll build foundational skills needed for any data analyst or business analyst role.

---

**Technical Foundation Required**

- Basic SQL syntax
- Aggregation functions (COUNT, SUM, AVG)
- Filtering with WHERE
- GROUP BY for summaries
- Currency formatting
- Case-based logic

Even if you're a beginner, this session is structured to teach everything from scratch. No prior data analytics experience is required.

---

**Time Investment**

4–5 hours of focused learning and practice.

You'll write real SQL queries, interpret business meaning, and prepare data summaries that mimic real-life analyst work.

---

**Business Context**

Imagine you're working for a fintech company (EduFin) that offers personal loans. A recent audit uncovered a discrepancy of ₹12 crores in the portfolio. Your task is to:

- Explore the structure and quality of customer and loan data

Identify gaps and inconsistencies

Calculate business metrics like total loan value, average loan, and risk distribution

Prepare executive summaries that decision-makers can trust

By the end of this session, you'll be ready to present real insights to leadership — using only SQL.

LEARNING OBJECTIVES

By completing this pretraining, you will master:

- ✅ SELECT statements with proper WHERE filtering
- ✅ COUNT, SUM, AVG, MIN, MAX aggregation functions
- ✅ GROUP BY for business categorization
- ✅ Percentage calculations using subqueries
- ✅ Professional Indian currency formatting
- ✅ Basic CASE statements for business logic
- ✅ NULL handling in financial data analysis
- ✅ Executive presentation formatting

## PART 1: DATA EXPLORATION

**Purpose:**

This section helps you get familiar with the structure and scale of the loan dataset. Before analyzing business performance, it's crucial to understand how your data looks, what each column means, and if any values are missing.

**Learning Goals:**

Preview data with basic SELECT statements.

Count rows to understand volume.

Identify missing or incomplete fields using NULL checks.

**-- Preview loan and customer data**

```
SELECT TOP 5 * FROM loans;

SELECT TOP 5 * FROM customers;
```

**-- Count records**

```
SELECT COUNT(*) as total_loan_records FROM loans;

SELECT COUNT(*) as total_customer_records FROM customers;
```

**-- Check missing disbursement dates**

```
SELECT
    COUNT(*) as total_records,
    COUNT(disbursement_date) as records_with_disbursement,
    COUNT(*) - COUNT(disbursement_date) as missing_disbursement_dates
FROM loans;
```

**-- Payments table completeness**

```
SELECT TOP 5 * FROM payments;

SELECT

    COUNT(*) as total_payments,

    COUNT(payment_amount) as payments_with_amount,

    COUNT(*) - COUNT(payment_amount) as missing_amounts

FROM payments;
```

## PART 2: FILTERING AND CLEANSING

**Purpose:**
Not all data is useful. Before performing analytics, filter out the records that are incomplete, invalid, or irrelevant. This ensures high-quality inputs for your analysis.

**Learning Goals:**

- Use WHERE clause to filter by valid values.
- Remove entries with missing key fields.
- Focus only on recent or relevant data based on business rules.

**-- Filter valid loan records**

```
SELECT loan_id, customer_id, loan_amount, loan_status

FROM loans

WHERE disbursement_date IS NOT NULL

  AND loan_amount > 0;
```

**-- Filter valid customers**

```
SELECT customer_id, customer_name, annual_income
FROM customers
WHERE annual_income IS NOT NULL
  AND registration_date >= '2023-01-01'
  AND city_id IS NOT NULL;
```

## PART 3: AGGREGATION MASTERY

**Purpose:**
Aggregation functions allow us to summarize large datasets into meaningful statistics like totals, averages, minimums, and maximums. These are the foundation for business reporting.

**Learning Goals:**

- Use SUM, AVG, MIN, MAX, and COUNT.
- Calculate basic business metrics like total portfolio size and average loan amount.
- Use conditional logic inside aggregation to break down portfolio status.

**-- Portfolio overview**

```
SELECT
    COUNT(*) as total_loans,
    SUM(loan_amount) as total_portfolio_value,
    AVG(loan_amount) as average_loan_size,
    MIN(loan_amount) as smallest_loan,
    MAX(loan_amount) as largest_loan
FROM loans
WHERE disbursement_date IS NOT NULL;
```

**-- Conditional aggregates**

```
SELECT
    COUNT(CASE WHEN loan_status = 'Active' THEN 1 END) as active_loans,
    COUNT(CASE WHEN loan_status = 'Defaulted' THEN 1 END) as defaulted_loans,
    SUM(CASE WHEN loan_status = 'Active' THEN loan_amount ELSE 0 END) as
active_portfolio
FROM loans
WHERE disbursement_date IS NOT NULL;
```

## PART 4: GROUPING AND CATEGORIZATION

**Purpose:**
GROUP BY is used to summarize data by categories like loan status or income type. This helps you derive insights like which type of loans are most common or risky.

**Learning Goals:**

- Group loans or customers by relevant business fields.
- Summarize each group with counts or averages.
- Use CASE inside GROUP BY for custom grouping like Small/Medium/Large loans.

**-- Group by loan status**

```
SELECT
    loan_status,
    COUNT(*) as loan_count,
    SUM(loan_amount) as category_total
FROM loans
WHERE disbursement_date IS NOT NULL
GROUP BY loan_status
ORDER BY loan_count DESC;
```

**-- Group loans by range**

```
SELECT
    CASE
        WHEN loan_amount < 500000 THEN 'Small Loans'
        WHEN loan_amount < 2000000 THEN 'Medium Loans'
        ELSE 'Large Loans'
    END as loan_category,
    COUNT(*) as count,
    AVG(loan_amount) as avg_amount
FROM loans
WHERE disbursement_date IS NOT NULL
GROUP BY
    CASE
        WHEN loan_amount < 500000 THEN 'Small Loans'
        WHEN loan_amount < 2000000 THEN 'Medium Loans'
        ELSE 'Large Loans'
    END
ORDER BY avg_amount DESC;
```

## PART 5: PERCENTAGE CALCULATIONS

**Purpose:**
Business leaders often want to know the *proportion* of values, not just raw totals. Calculating percentages lets you compare across categories.

**Learning Goals:**

- Derive percentages using subqueries and math operations.
- Present loan type or city-wise distributions as a percentage of the total.
- Round results for clean, readable reporting.

**-- Loan count percentage**

```
SELECT
    loan_status,
    COUNT(*) as loan_count,
    ROUND(
        COUNT(*) * 100.0 / (
            SELECT COUNT(*)
            FROM loans
            WHERE disbursement_date IS NOT NULL
        ),
        2
    ) as percentage_of_portfolio
FROM loans
WHERE disbursement_date IS NOT NULL
GROUP BY loan_status
ORDER BY loan_count DESC;
```

**-- Loan amount percentage**

```
SELECT
    loan_status,
    SUM(loan_amount) as status_amount,
    ROUND(
        SUM(loan_amount) * 100.0 / (
            SELECT SUM(loan_amount)
            FROM loans
            WHERE disbursement_date IS NOT NULL
        ),
        2
    ) as percentage_of_value
FROM loans
WHERE disbursement_date IS NOT NULL
GROUP BY loan_status;
```

## PART 6: INDIAN CURRENCY FORMATTING

**Purpose:**
Financial reports should follow the Indian currency format (₹, lakhs, crores). Using proper formatting makes your analysis professional and executive-friendly.

**Learning Goals:**

- Use FORMAT() to convert numbers into currency.
- Convert raw numbers into lakhs/crores using math and string formatting.
- Apply localized presentation standards (like '₹ 25,00,000' instead of '2500000').

**-- Indian formatting**

```
SELECT
    loan_status,
    COUNT(*) as loan_count,
    FORMAT(SUM(loan_amount), 'C0', 'en-IN') as portfolio_value_formatted,
    FORMAT(AVG(loan_amount), 'C0', 'en-IN') as average_loan_formatted
FROM loans
WHERE disbursement_date IS NOT NULL
GROUP BY loan_status;
```

**-- Format as Crores**

```
SELECT
    loan_status,
    COUNT(*) as loans,
    CONCAT('₹', FORMAT(SUM(loan_amount)/10000000, 'N2'), ' Cr') as portfolio_crores
FROM loans
WHERE disbursement_date IS NOT NULL
GROUP BY loan_status;
```

## PART 7: EXECUTIVE SUMMARY REPORTING

**Purpose:**
Executives expect ready-to-use dashboards or reports. Your queries should produce data that's clean, categorized, formatted, and insightful in one view.

**Learning Goals:**

- Combine aggregates, percentages, and formatting into one result.
- Use aliases (AS) for clean column names.

- Sort output to highlight highest value or risk categories.

**-- Decision-ready summary**

```
SELECT
    loan_status as 'Loan Status',
    COUNT(*) as 'Number of Loans',
    FORMAT(SUM(loan_amount), 'C0', 'en-IN') as 'Portfolio Value',
    CONCAT(
        ROUND(COUNT(*) * 100.0 / (
            SELECT COUNT(*) FROM loans WHERE disbursement_date IS NOT NULL
        ), 1), '%'
    ) as 'Portfolio %'
FROM loans
WHERE disbursement_date IS NOT NULL
GROUP BY loan_status
ORDER BY SUM(loan_amount) DESC;
```

## PART 8: CASE LOGIC – INCOME OR RISK CATEGORIES

**Purpose:**
CASE statements allow you to create custom categories or apply business rules directly in SQL. This makes your analysis more intelligent and actionable.

**Learning Goals:**

- Create income tiers (Low, Medium, High) or risk levels.
- Tag records based on business-defined thresholds.
- Assign actions or recommendations for each category.

**-- Income categories**

```
SELECT
    customer_id,
    annual_income,
    CASE
        WHEN annual_income < 300000 THEN 'Low Income'
        WHEN annual_income < 800000 THEN 'Medium Income'
        WHEN annual_income < 1500000 THEN 'High Income'
        ELSE 'Premium Income'
    END as income_category
FROM customers
WHERE annual_income IS NOT NULL;
```

**-- Business actions**

```
SELECT
    loan_status,
    COUNT(*) as loan_count,
    CASE
        WHEN loan_status = 'Defaulted' THEN 'URGENT: Immediate Action Required'
        WHEN loan_status = 'Overdue' THEN 'HIGH PRIORITY: Collection Focus'
        WHEN loan_status = 'Active' THEN 'STANDARD: Regular Monitoring'
        ELSE 'REVIEW: Status Verification Needed'
    END as business_action
FROM loans
WHERE disbursement_date IS NOT NULL
GROUP BY loan_status;
```

## PART 9: MULTI-LEVEL BUSINESS LOGIC

**Purpose:**
Business problems often require more than one condition. Nested and multiple CASE statements help encode multi-level logic, like risk + recommendation.

**Learning Goals:**

- Use CASE for layered decision making.
- Build logic chains (IF-ELSE-like) within SQL.
- Show both assessment and recommendation in one query.

**-- Risk and processing recommendation**

```
SELECT
    customer_id,
    annual_income,
    CASE
        WHEN annual_income IS NULL THEN 'CRITICAL: Missing Income Data'
        WHEN annual_income < 200000 THEN 'HIGH RISK: Low Income Segment'
        WHEN annual_income BETWEEN 200000 AND 500000 THEN 'MEDIUM RISK: Moderate Income'
        WHEN annual_income BETWEEN 500001 AND 1000000 THEN 'LOW RISK: Stable Income'
        ELSE 'PREMIUM: High Income Segment'
    END as risk_assessment,
    CASE
        WHEN annual_income < 300000 THEN 'Enhanced Documentation Required'
        WHEN annual_income > 1500000 THEN 'Fast Track Eligible'
        ELSE 'Standard Processing'
    END as processing_recommendation FROM customers;
```

## PART 10: DATA QUALITY AND VALIDATION

**Purpose:**
Bad data = bad analysis. Before presenting or using data, check for missing, invalid, or illogical entries. SQL helps automate these validations.

**Learning Goals:**

- Detect NULLs, future dates, or negative values.
- Use COALESCE() for fallback/default values.
- Present a validation summary for quality assurance.

**-- Missing values check**

```
SELECT
    'Customers' as table_name,
    COUNT(*) as total_records,
    COUNT(customer_name) as valid_names,
    COUNT(phone_number) as valid_phones,
    COUNT(annual_income) as valid_incomes,
    COUNT(*) - COUNT(customer_name) as missing_names
FROM customers
UNION ALL
SELECT
    'Loans' as table_name,
    COUNT(*) as total_records,
    COUNT(disbursement_date) as valid_disbursements,
    COUNT(loan_status) as valid_status,
    COUNT(loan_amount) as valid_amounts,
    COUNT(*) - COUNT(disbursement_date) as missing_disbursements
FROM loans;
```

**-- Safe default handling**

```
SELECT
    customer_id,
    COALESCE(annual_income, 0) as income_for_calculation,
    COALESCE(phone_number, 'Contact Missing') as contact_status
FROM customers;
```

**-- Business rule validation**

```
SELECT
    'Invalid Loan Amounts' as issue_type,
    COUNT(*) as problem_count
FROM loans
WHERE loan_amount <= 0 OR loan_amount IS NULL
UNION ALL
SELECT
    'Future Disbursement Dates' as issue_type,
    COUNT(*) as problem_count
FROM loans
WHERE disbursement_date > GETDATE()
UNION ALL
SELECT
    'Missing Customer Data' as issue_type,
    COUNT(*) as problem_count
FROM customers
WHERE customer_name IS NULL OR customer_name = '';
```